CSC 4103 - Operating Systems
Spring 2008

LECTURE - XI
MAIN MEMORY

Tevfik Koşar

Louisiana State University
February 28th, 2008

Memory Management Requirements

➢ **The O/S must fit multiple processes in memory**

✓ memory needs to be subdivided to accommodate multiple processes

✓ memory needs to be allocated to ensure a reasonable supply of ready processes so that the CPU is never idle

✓ memory management is an **optimization** task under **constraints**
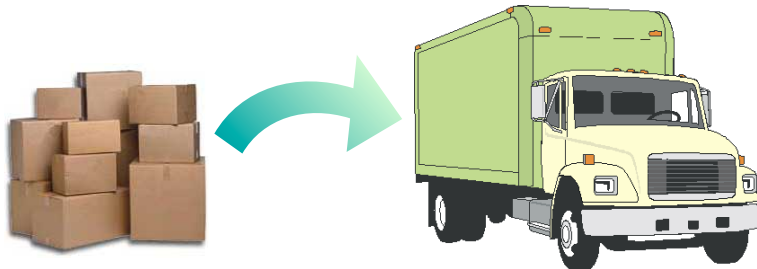
Fitting processes into memory is like fitting boxes into a fixed amount of space

Memory Allocation - contiguous

- Fixed-partition allocation
  - Divide memory into fixed-size partitions
  - Each partition contains exactly one process
  - The degree of multi programming is bound by the number of partitions
 - When a process terminates, the partition becomes available for other processes

  ➔ no longer in use

OS
process 5
process 9
process 10
process 2

Memory Allocation (Cont.)

- Variable-partition Scheme
  - When a process arrives, search for a hole large enough for this process
  - Hole – block of available memory; holes of various size are scattered throughout memory
  - Allocate only as much memory as needed
  - Operating system maintains information about:
    a) allocated partitions   b) free partitions (hole)

OS
process 5
process 2

⇒

OS
process 5
process 9
process 2

⇒

OS
process 5
process 9
process 10
process 2
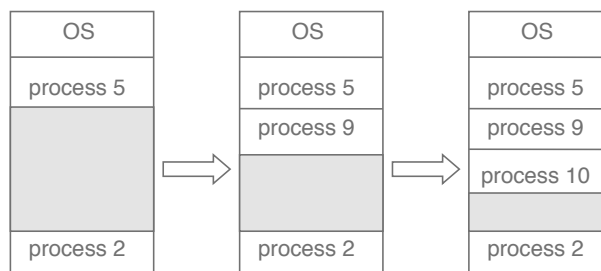
Dynamic Storage-Allocation Problem

How to satisfy a request of size *n* from a list of free holes

- **First-fit**: Allocate the *first* hole that is big enough
- **Best-fit**: Allocate the *smallest* hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
- **Worst-fit**: Allocate the *largest* hole; must also search entire list. Produces the largest leftover hole.

First-fit and best-fit better than worst-fit in terms of speed and storage utilization

Example

Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)? Which algorithm makes the most efficient use of memory?

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous (in average ~50% lost)
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time

15

Logical Address Space

- Each process has a separate memory space
- Two registers provide address protection between processes:
- Base register: smallest legal address space
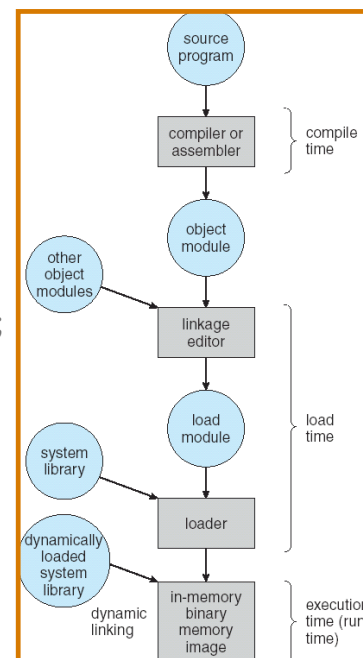- Limit register: size of the legal range
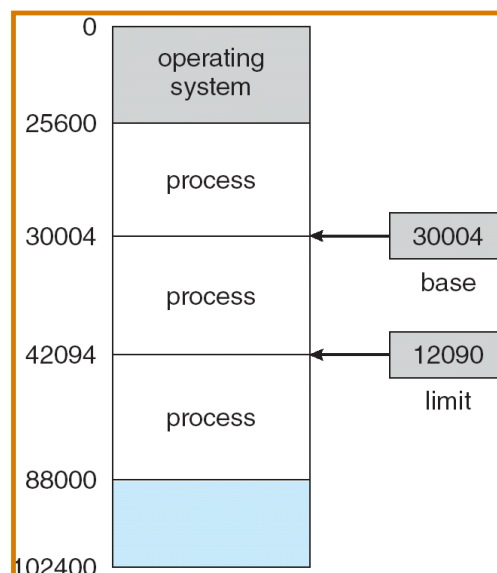
8

Address Binding

- Addresses in a source program are generally symbolic
    - eg. int count;
- A compiler binds these symbolic addresses to relocatable addresses
    - eg. 100 bytes from the beginning of this module
- The linkage editor or loader will in turn bind the relocatable addresses to absolute addresses
    - eg. 74014
- Each binding is mapping from one address space to another

9

Binding of Instructions and Data to Memory

Address binding of instructions and data to memory addresses can happen at three different stages

- **Compile time**: If memory location known a priori, *absolute code* can be generated; must recompile code if starting location changes
- **Load time**: Must generate *relocatable code* if memory location is not known at compile time; need only reload if starting address changes
- **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps.

source program

compiler or assembler — compile time

object module

other object modules

linkage editor

load module — load time

system library

loader

dynamically loaded system library

dynamic linking

in-memory binary memory image — execution time (run time)

10

Logical vs Physical Address Space

- Address generated by CPU is referred as **logical address**
- Address seen by the memory is seen as **physical address**
- Compile time and load time methods generate identical logical and physical addresses
- Execution-time method generates different logical and physical addresses
 - Logical address referred as **virtual address**

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address

- In MMU scheme, the value in the **relocation register** (base register) is added to every address generated by a user process at the time it is sent to memory

- The **user program** deals with *logical* addresses; it **never sees** the *real* physical addresses
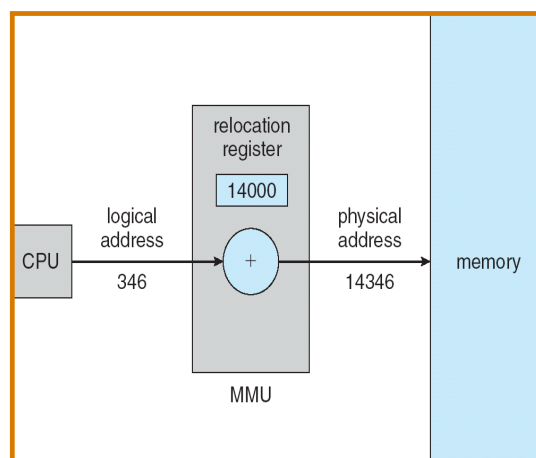
HW Address Protection

- CPU hardware compares every address generated in user mode with the registers
- Any attempt to access other processes' memory will be trapped and cause a fatal error
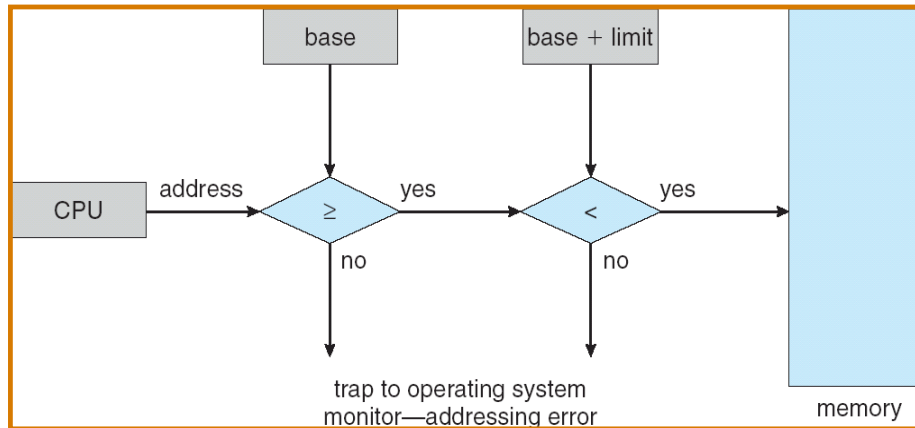
Paging - noncontiguous

- Physical address space of a process can be noncontiguous
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8192 bytes)
- Divide logical memory into blocks of same size called **pages**.
- Keep track of all free frames
- To run a program of size n pages, need to find n free frames and load program
- Set up a page table to translate logical to physical

Address Translation Scheme

- Address generated by CPU is divided into:

  – *Page number (p)* – used as an index into a *page table* which contains base address of each page in physical memory

  – *Page offset (d)* – combined with base address to define the physical memory address that is sent to the memory unit
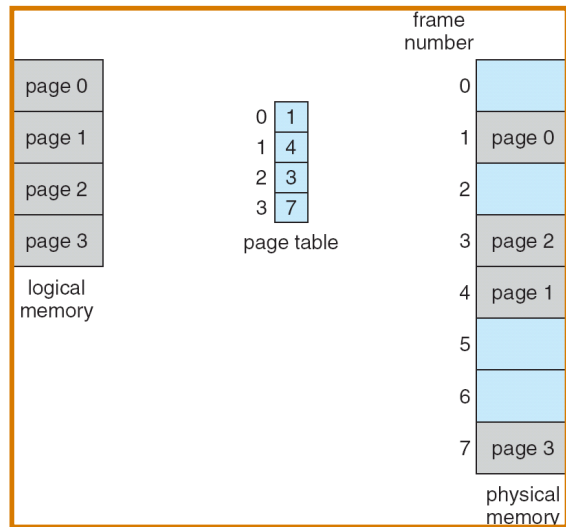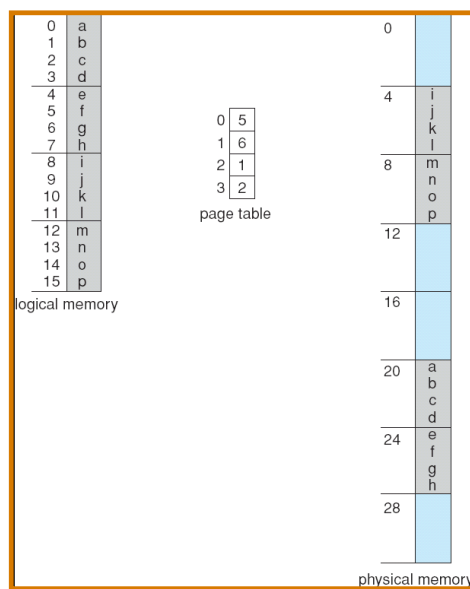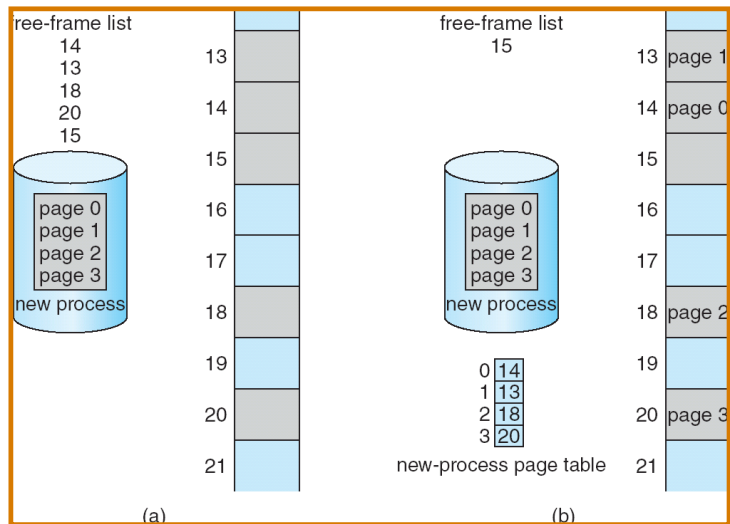
Address Translation Architecture
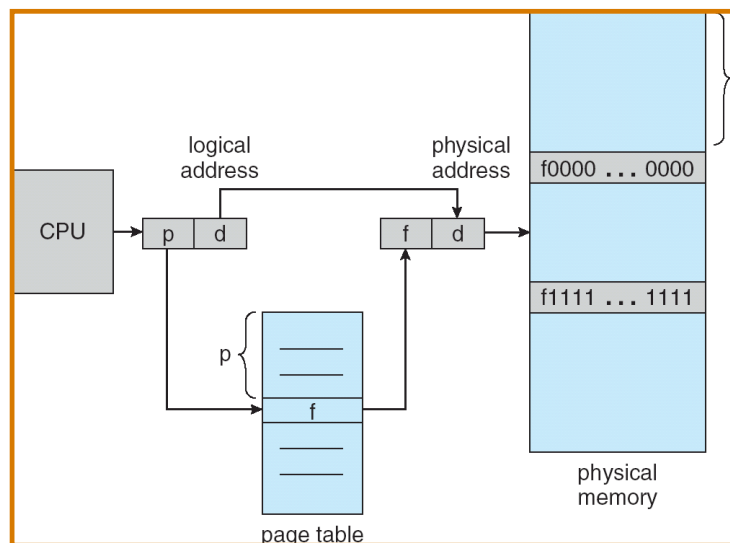
Paging Example

Paging Example

Free Frames

Shared Pages

- **Shared code**
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  - Shared code must appear in same location in the logical address space of all processes

- **Private code and data**
  - Each process keeps a separate copy of the code and data
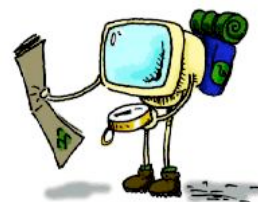 - The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example
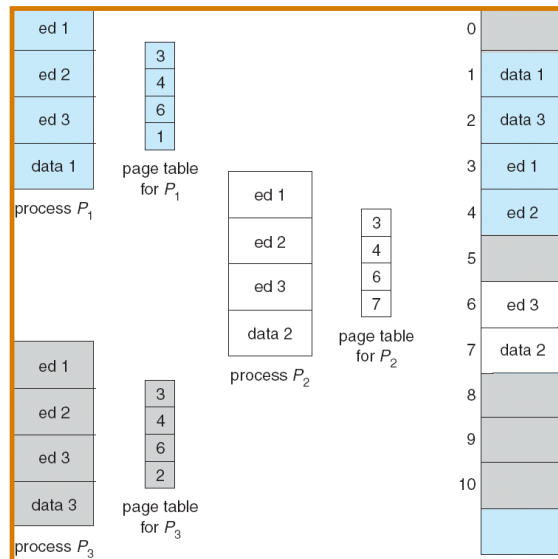
Midterm

- Chapters 1 - 7 from Silbershatz
- Important Topics:
  - Processes
  - Threads
  - CPU Scheduling
  - Process Syncronization
  - Threads
- Closed book exam
- Midterm review next Tuesday
- Solve the sample exam